BCM2550 – Automne 2010

Cours #1 : Introduction à UNIX et Perl

Le but de ce travail pratique sera de vous familiariser premièrement avec votre environnement de travail, soit le system UNIX et dans un second temps, vous familiariser avec le langage de programmation que nous utiliserons pour l'ensemble de ce cours.

1A – Connexion à l'environnement ESIBAC:

- Ouvrir le répertoire Logiciels spécialisés
- Ouvrir le répertoire Hummingbird
- Double-cliquer sur xdm_esilbac1.xs
- Saisir Nom d'utilisateur et Mot de passe
- Cliquer sur le bouton Connexion

1B – Introduction à l'environnement Linux SUSE

Un certain nombre de logiciels sont disponibles sur l'environnement tel qu'Open Office, Firefox, etc. Pour lancer des applications par une ligne de commande, ouvrir un **Terminal** ou **Console** à partir du 3eme bouton en bas à gauche (**Ecran+Coquillage**).

2 – Introduction aux différentes commandes Linux

Dans cette section seront décrites certaines des lignes de commandes les plus utilisées sur Linux. Ces lignes de commandes vous serviront entres autres à vous déplacer d'un répertoire à l'autre, afficher ce qu'il y a dans un répertoire, copier un fichier, afficher le contenu d'un fichier, modifier le contenu d'un fichier, etc.

- La commande ***pwd*** : sert à voir dans quel répertoire, ou chemin de l'arborescence vous vous situez.

- La commande ***ls*** : sert à voir ce qu'il y a dans votre répertoire.

- La commande ***cd*** : pour « **c**hange **d**irectory », nous permet de changer de répertoire.
  - *cd* nom_du_répertoire

- La commande ***mkdir*** : nous permet de créer un nouveau répertoire.

- *mkdir* nouveau_répertoire

- La commande ***rmdir*** : nous permet d'effacer un dossier vide.
  - *rmdir* dossier_à_effacer

- La commande ***cp*** : nous permet de copier un fichier ou un dossier.
  - *cp* fichier1 fichier2
  - *cp –r* dossier1 dossier2
  - *cp* fichier1 dossier1/ #Copie le fichier1 dans le dossier1
  - *cp* fichier1 dossier1/fichier2 #Idem sauf renomme fichier1

- La commande ***mv*** : nous permet de déplacer un fichier ou un dossier.
  - Même syntaxe que pour *cp*

- La commande ***man*** : nous permet de voir le manuel d'utilisation pour une autre commande.
  - *man* ls
  - *Pour quitter le manuel, appuyez sur 'q'*

- La commande ***more*** : nous permet de voir et naviguer dans le contenu d'un fichier.
  - *less* fichier1
  - *Pour quitter, appuyez sur 'q'*

- La commande ***gzip*** nous permet d'archiver un fichier ou un dossier et la commande ***gunzip*** nous permet de les désarchiver.
  - *gzip* fichier #Attention votre fichier original devient *fichier.gz*
  - *gunzip* fichier.gz

- La commande ***tar*** : permet de créer un fichier archivé de plusieurs fichiers et/ou dossiers.
  - *tar –cvf* fichiers_archivés.tar fichier1 dossier1
  - *gzip* fichiers_archivés.tar #Devient fichiers_archivés.tar.gz
  - *tar –czvf* fichiers_archivés.tar.gz fichier1 dossier1
    #Combine les deux premiers.
  - Pour décompresser :
    - *tar* –xvf fichiers_archivés.tar
    - *tar* –xzvf fichiers_archivés.tar.gz

- Un éditeur de texte très bien et pas trop complexe qui vous permettra d'écrire vos programmes : ***gedit***
  - Pour ouvrir un fichier texte :
    - *gedit* fichier1 &
    - Le signe « & » vous permet de pouvoir continuer à vous servir de votre terminal lorsqu'un autre programme est en marche.

- La commande ***chmod*** vous servira à changer les différentes permissions pour exécuter ou permettre à d'autres utilisateurs que vous de voir ou de modifier un fichier.
  - *chmod ugo+rwx* fichier1
  - u = utilisateur, g = groupe, o = other, + ou - = ajouter ou enlever, r = droits de lecture, w = droits d'écriture, x = droits d'exécution
  - Pour vos scripts, il vous faudra autoriser l'exécution : *chmod u+x*

3 – Introduction à Perl

Maintenant que vous savez comment utiliser votre environnement de travail, nous pouvons aborder les choses sérieuses. Cette section sera une introduction afin que vous soyez capables de faire un script Perl de base.

Afin de lancer un script Perl, il vous faut soit changer les permissions du fichier script ou le lancer de la façon suivante :

%> perl script.pl   #Où script.pl est le nom du script

De bonnes habitudes à prendre dès le départ en programmation sont de **sauvegarder** souvent vos travaux, faire des **copies de sauvegarde** fréquemment au cas où vous feriez des modifications fâcheuses qui occasionnent de nouveaux problèmes que vous n'aviez pas et de commenter son code (signe #, suivi des commentaires).
Ex :

```
#Mon premier commentaire…
```

Il est important pour indiquer à votre système qu'il s'agit d'un script Perl de mettre un en-tête dans votre fichier script (à la première ligne):

```
#!/usr/bin/perl -w
```

Cet en-tête indique au système le chemin à prendre pour aller trouver l'interpréteur de code.

L'impression à l'écran et les variables.

Comme premier exercice, commencez par faire une impression à l'écran avec la commande *print*.

```
print "Bonjour le monde!";
print "Bonjour ","le ","monde!";
print "Bonjour " . "le " . "monde!";
```

Vous pouvez aussi créer une variable scalaire et imprimer celle-ci à l'écran avec la commande *print*.

```
$dna = "CATTAG";
print $dna;
```

Si vous voulez jouer avec des chiffres, Perl interprétera les lignes de commandes que vous lui donnerez.

```
$a = "5";
$b = $a + 4;
print $b; # Vous imprimera 9

$c = $a . 4;
print $c; # Vous imprimera 54

$d = "cent";
print "$a$d"; #Vous imprimera 5cent
```

Le caractère "\n" vous permet de faire un saut de ligne lors de l'impression. Ainsi :

```
print "Bonjour" . "le" . "monde!";
```

Donnera

Bonjourlemonde!

et

```
print "Bonjour\n" . "le\n" . "monde!\n";
```

Donnera

Bonjour
le
monde!

Une fonction utile lorsqu'on travaille avec des chaînes de caractères (string) est la commande `length()` qui nous donne la taille de la chaîne mise entre parenthèses.

```
print length("Bonjour"); #Imprimera la valeur 7
```

Si vous êtes à la recherche d'une ligne de commande fautive, les commandes print ainsi que la commande *`exit;`*** peuvent vous être très utiles.

Les conditions if, else

Voici la structure pour faire un test de conditions.

```
if(condition){
      Faire l'opération;
}
else{
      Faire l'opération;
}
```

Les différents symboles utilisés pour réaliser des tests:
(`==, !=, <, >, <=, >=`) #Pour les chiffres
(`gt, lt, ge, le, eq`) #Pour les chaînes de caractères
Les signes logiques vous permettent de voir plusieurs conditions différentes à la fois.

```
&& # Opérateur "et"
|| # Opérateur "ou"
```

Exemple :

```
print "quelle est la date de naissance de mozart\n";
print "entrez une année : ";
$annee = <STDIN>;
print "entrez un mois : ";
$mois = <STDIN>;
if (($annee == 1756) && ($mois eq "janvier"))  {
      print "bravo!\n";
}
else{
      print "erreur\n";
}
if (($annee == 1756) || ($mois eq "janvier"))  {
      print "au moins l'une des deux réponses était correcte\n";
}
else  {
      print "vraiment tout faux!\n";
}
```

Faites plusieurs exemples afin de bien comprendre ces opérateurs, ils seront très importants!

Commande de substitution

Il est très facile en Perl de faire des changements dans une chaîne de caractères. Il s'agit d'établir une nouvelle variable avec la commande de substitution. Voici comment l'utiliser :

```
$nouvelle_variable =~ s/"Ancien"/"Nouveau"/g;
```

Comme exercice, changez cette séquence d'ADN en séquence d'ARN.

```
$adn = "CATGTCATTG";
```

Prendre des noms de fichiers en argument

Nous voudrions lancer un script de la façon suivante :

%> mon_script.pl <fichier_entrée> <fichier_sortie>

Pour cela, il faudra être capable d'aller chercher les valeurs mises en entrée. Ces valeurs sont mises dans une variable spéciale par Perl. Cette variable se nomme `@ARGV`. C'est un tableau (nous verrons les tableaux plus tard). Afin de ne pas avoir de problèmes pour notre exécution, nous devrons gérer les exceptions ainsi que les mauvaises utilisations du script. Il faudra éviter d'avoir trop ou peu d'arguments. Voici comment :

```
if( @ARGV != 2 ){ # tester le nombre d'arguments donnes
      die "erreur: il faut 2 arguments\n";
}
$fichier_entree = $ARGV[0]; # premier argument
$fichier_sortie = $ARGV[1]; # second argument
```

L'entrée standard (SDTIN).

L'entrée standard se lit ligne paar ligne:

```
$seq = <STDIN>; # 1 ligne de l'entree standard est lue et stockée
dans la variable $seq
```

Attention à bien enlever le retour de chariot:

```
chomp $seq;
```

Manipulation de fichiers

Lecture de fichier

La manière la plus simple d'utiliser des fichiers externes et de les lire se fait avec deux commandes. Une sert à ouvrir le fichier : `open(FICHIER,"fichier")` et l'autre à le fermer : `close(FICHIER)`. Bien sûr, "fichier" peut être remplacé par une variable. De plus,

```
open(ENTREE,"$fichier_entree") or die "Impossible d'ouvrir le
fichier!\n"; #Si le fichier n'existe pas, le programme quitte.

$seq = <ENTREE>; # 1 ligne du fichier est lue et stockée dans $seq

close(ENTREE);
```

Écriture dans un fichier

Si nous voulons écrire dans un fichier, il faut ajouter le caractère d'écriture > :

```
open(SORTIE,">$fichier_sortie");
```

Pour écrire dans un fichier, il s'agit de faire la commande `print`, mais avec le nom du fichier dans lequel nous voulons écrire tout juste après le mot "`print`" :

```
print SORTIE "CATAGGA\n";
```